Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 93 (2016) 799 – 807

Procedia Computer Science

6th International Conference On Advances In Computing & Communications, ICACC 2016, 6-8 September 2016, Cochin, India

Towards Improving Storage Cost and Security Features of Honeyword Based Approaches

Nilesh Chakraborty*, Samrat Mondal

Indian Institute of Technology Patna, Patna, 801103, India

Abstract

Password based authentication shows its vulnerability against inversion attack model in which adversary obtains plaintext password from its corresponding hashed value. To cope up with such attack, honeyword based authentication technique is introduced. In this technique, along with the original password of user, some dummy passwords or honeywords are also stored. Although this technique is good enough to address the aforementioned security breach, but use of additional storage to store the honeywords is still an overhead associated with such approach. In this paper, we have proposed few directions to minimize the storage cost of some of the existing honeyword generation approaches. We have even found that in some cases no additional storage overhead is required. A comparative analysis at the end also shows that the proposed techniques are able to raise some of the security features compared to existing honeyword generation approaches.

Peer-review under responsibility of the Organizing Committee of ICACC 2016

Keywords: Security; Password; Honeyword; Inversion Attack; Storage Cost.

1. Introduction

There are mainly four factors related to user authentication: authentication by something user knows (e.g., password), authentication by something user has (e.g., physical token), authentication by something the user is (e.g., biometric authentication) and authentication by someone user knows[1]. Among these four factors of authentication techniques, password based authentication is widely accepted for its simple login functionality and ease of memorability. Because of its popularity, password based authentication schemes have also been explored using different attack models[2 3 4]. Inversion attack is one such lately developed attack model[5] on password based authentication technique which opens up a new security breach for those systems which are protected by password.

* Corresponding author. Tel.: +919472472513;
E-mail address: nilesh.pcs13@iitp.ac.in

Peer-review under responsibility of the Organizing Committee of ICACC 2016
doi:10.1016/j.procs.2016.07.298

1.1. Inversion attack

System generally stores usernames and passwords in password file (F_P). Though usernames are stored in the plain text, passwords are maintained in hash format. If F_P is compromised, an adversary readily gets the data stored in F_P. Retrieving user's password from the hashed value is the main objective of inversion attack model. Earlier adversary performed brute force search to obtain the user's password from the corresponding hashed password. In 2008[6], Jhon the Ripper developed a technique based on password reuseability policy of users on the dictionary words which helped to reduce the brute force search complexity with high percentages. In 2009, Weir et al.[7] proposed an algorithm based on probabilistic context free grammar to further improve the inversion rate from 28% to 129%. Recently in 2014, Ma et al. showed that by using hidden markov chain model, system can invert the hashes with higher success rates[5]. Thus, by performing inversion attack, adversary may obtain user password (p) from corresponding hashed value (H(p)).

There are some significant evidences of inversion attack on some reputed web based organizations. Report reveals that adversaries have successfully inverted many passwords of some giant web based service provider like Yahoo, LinkedIn, RockYou and others[8]. In recent past, 50 million passwords of Evernote have been successfully inverted under the same attack scenario[9]. Thus, there is a strong urgency of developing a security model which robustly handles this attack.

1.2. Honeyword based authentication technique: a security measure against inversion attack

In honeyword based authentication technique, against each username, system maintains *k-1* ($k > 2$) dummy passwords (also known as honeywords) along with original user password. Index position (c_i where $1 \leq c_i \leq k$) of original password is stored in a different server, known as honeyChecker and is denoted here as H_C. From the adversary perspective, after compromising the F_P, she gets confused among k passwords. If adversary picks a honeyword then directed index value to H_C doesn't get matched with the original password index, stored in H_C and thus, breach is detected. Therefore, honeyword based authentication technique lures the adversary with a probability value of $\frac{k-1}{k}$.

Though honeyword based authentication technique gives tremendous support to get rid off the inversion attack but there exists a inherit drawback of all honeyword based approaches proposed so far. Using a standard hashing algorithm (e.g. SHA−1) system generates h bytes long hashed string. Thus, honeyword based approaches incur an additional storage overhead of (k-1) $\times$ h bytes per account, which is a serious concern[10,11].

1.3. Motivations and Contributions

Reducing the storage overhead of some newly proposed honeyword generation methods (which overcome the limitations of previously proposed schemes) is our primary motivation behind this work. Along with reducing storage overhead we have also tried to improve the security standard without degrading the usability standard of the existing techniques. Mainly driven by these factors we have made following contributions in this paper.

- **Contribution 1:** We have considered four existing honeyword generation techniques − (a) Modified-tail (b) Caps-key (c) Close-number-formation and (d) Pre-processing to minimize their storage overhead.
- **Contribution 2:** The security standard of those schemes are also been improved during the course of optimizing the storage cost.

2. Security, Usability and Storage Standards of Honeyword Based Schemes

To begin with, we first discuss the security standard of honeyword based authentication techniques and followed by this we introduce the usability standard and storage overhead of honeyword based approaches. Security, storage and usability standards are collectively referred as *SSU* standards or, *SSU* parameters in the subsequent sections.

2.1. Security Standards

The security standard of any honeyword based approach can be measured with respect to three parameters (a) DoS resiliency (b) Multiple System Vulnerability (MSV) and (c) Flatness. These parameters are elaborated next.

(a) DoS resiliency: Knowing the original password of a user if adversary can guess any honeyword, maintained for that user acount, then adversary may intentionally submit the honeyword to raise an alarm by the H_C. In this scenario, system assumes that F_P has been compromised when it is actually not. This phenomenon is known as DoS attack. *Chaffing-by-tweaking*[10] methods are identified as weak DoS resilient schemes where the generated honeywords sometimes are easily guessable by the adversary. On the other hand *modeling-syntax-approach*[12] provides strong security against such attack.

(b) Multiple System Vulnerability For a given password, a honeyword generation algorithm creates different list of honeywords at each run. Thus, using same honeyword generation algorithm, two different systems produce different lists of honeywords for the same user password. If adversary becomes able to compromise both these F_Ps then by performing intersection operation between those two lists of passwords, the intruder gets the original password of the user. This is known as multiple system vulnerability of honeyword generation approaches. Most of the honeyword based algorithms provide weak security against MSV. However, *take-a-tail*[10], paired-distance-protocol[13] are two exceptions in this context.

(c) Flatness: Flatness property of a honeyword based approach ensures that all the sweetwords (honeywords along with user's original password) stored against a username should be equally probable from the adversary perspective while selecting the original user's password (or, sugarword). In other words, using a perfectly flat honeyword generation approach, adversary has no advantages in terms of identifying the original user's password. A good honeyword generation approach requires to be perfectly flat (e.g., *take-a-tail*) to lure the attackers among k possible sweetwords.

2.2. Usability Standards

The usability standard of a honeyword generation approach is determined through three parameters namely − (a) System interference (b) Stress on memorability and (c) Typo safety.

(a) System interference A honeyword generation approach is said to have system interference if it forces users to remember some extra information during login along with username and password. *Take-a-tail*[10] approach interferes in password choice of users whereas *modeling-syntax-approach*[12] has no system interference.

(b) Stress on memorability: Stress on memorability is directly related to system interference parameter. Remembering additional information imposes extra load on human mind. Depending on how much extra information is to be remembered by user, this feature can be classified into two categories (i) High stress on memorability: where (e.g. take-a-tail) user remembers n system generated information as a part of his login credential for n different web accounts. (ii) Low stress on memorability: where (e.g. caps-key) user may remember no/single extra information of his own choice to login into n different accounts. A good honeyword system always imposes low stress on memorability to maintain a high usability standard.

(c) Typo safety: A highly typo safe honeyword system ensures that typing mistakes of user, during the course of entering password, hit a honeyword with negligible probability. This is a very important criterion from the usability perspective as a less typo safe method can lead to a false detection and block the user account due to typing mistake of a genuine user. *Chaffing-by-tweaking* methods are less typo safe whereas *modeling-syntax-approach* sets high typo safety.

2.3. Storage Cost

The passwords are stored in the system database after converting into an equivalent hashed form. If the hashed strings is considered as h bytes long, then to store single password information h bytes memory space will be required. On the other hand as honeyword based authentication technique stores k sweetwords to lure the attackers thus, the required storage cost becomes $k \times h$ bytes. In[10] authors argued that suitable value of k should be 20 to maintain a moderate detection rate. Using a standard hashing algorithm like SHA−256 the values of h becomes 32 bytes. So honeyword based authentication technique incurs an additional storage overhead of (k-1) $\times$ h or, $19 \times 32 = 608$ bytes. This can be identified as one of the significant drawbacks of honeyword or decoy password generation protocols[10,11].

3. Review of Some Recently Proposed Approaches

As discussed earlier, while selecting few honeyword generation approaches under consideration, we focus on recently proposed approaches in [14], as those overcome most of the limitations of previously proposed approaches in this domain. Under the scope of this section we give brief overviews on proposed methods in [14].

3.1. Modified−tail

In this approach user selects extra information (*tail*) of length ℓ from a set (S) of characters as a part of his login credential. The set of characters are chosen in such a manner so that they avoid few obvious patterns (e.g. sequential key stroke) in the keyboard. In Fig. 1, we show a probable list of elements that can be chosen to form the set S as shown in [14].

Fig. 1. Elements @ | and ? are selected as members of S

The honeywords are generated from the different combinations of elements from S, having length $|S|$, appended to the user password. For example, if user selects his password as *alex* and selects tail as ?@ then the list of sweetwords for the set S are presented below −

alex?|@ **alex?@|** alex@?| alex@|? alex|?@ alex|@?

One interesting characteristic of *modified-tail* is that from a set of $|S|$ characters, user has to choose $|S|$-1 characters as tail. While verifying the password, system appends the remaining character to the tail to construct the complete password. Thus, if user enters *alex?@*, where ?@ is the selected tail by user, then system appends the remaining character '|' from S to the entered string and treats *alex?@|* as user's submitted password.

Reviewing the *SSU* parameter of modified-tail: As *modified−tail* demands an extra information to be remembered by the user of his own choice thus, stress on memorability using this approach is on lower side with having system interference. The method is also typo safe as selected characters to build the set S are far apart from each other on a standard keyboard (as shown in Fig. 1). This method maintains high security standard for MSV and Flatness parameters. *Modified−tail* is a weak DoS resilient scheme thus, light security policy against DoS may be adopted. This method requires $k \times h$ bytes of storage cost to setup the inversion attack detection architecture.

3.2. Caps-key

Using *caps-key* based honeyword generation approach, user selects m alphabets in capital from his password containing n (where n > m) alphabets. From the chosen password of user, system generates the honeywords after taking different combinations of m alphabets in capital. Thus, number of possible sweetwords by using this approach becomes $\binom{n}{m}$. Among the possible options of sweetwords, system chooses k sweetwords according to its requirement. For example, for the password as *AnImal* − where user chooses $m = 2$ letters in capital, there are $\binom{6}{2} - 1 = 14$ possible ways for generating the honeywords. Now for $k = 6$, system may generate following list of sweetwords −

AniMal aNImal **AnImal** animAL AnimaL animal

Reviewing the SSU parameter of caps-key : Using this approach, as user requires to remember some extra information (in terms of capital letters in password) of his own choice to cope up with the honeyword based scheme thus, the method imposes low stress on memorability with some system interference. The method is also less typo safe as it might be case that users typing mixed case passwords will slow down to press the caps lock or shift key

or it might be that they will type at the same speed and suffer from a sticky shift or too soon/late presses that will increase their typo rate. From the security perspective this method generates absolute flat honeywords. The security parameters related to this approach have also been re-analysed here.

In [14], authors shows that for a fixed value of m, security against MSV depends on n. As n increases, for fixed value of both m and k, security against MSV decreases. In [14] authors also mentioned the following relation between DoS and MSV parameters –

$$\Pr(\text{DoS}) \propto \frac{1}{\Pr(\text{MSV})} \tag{1}$$

Thus, with the increasing value of n, security against DoS goes high using this method. Storage cost for maintaining sweetwords using this approach can be evaluated as same as *modified-tail* approach.

3.3. Close-number-formation

Using *close-number-formation* honeyword based algorithm, system generates decoy passwords that are much closed to the number, used in the original password. For example, if original password chosen by user contains 1992 (e.g. *alex1992*) then formed sweetwords for $k = 6$ may be as follows –

alex1994 alex1996 alex1999 **alex1992** alex1990 alex1989

For generating close numbers, system maintains two sets as $num = \{1, 2, 3\}$ and $sig = \{+, -\}$. After retrieving the digit (*dp*), used in the user password, system generates the digit (*dh*) used in first honeyword, by tweaking *dp* using following equation.

$$\text{dh} = \text{dp} \quad \text{e(sig)} \quad \text{e(num)} \tag{2}$$

Here *e(sig)* and *e(num)* denote an element from the set *sig* and *num*, respectively. For generating subsequent honeywords, system randomly selects a digit from the generated sweetwords. After selecting the digit, system tweaks that digit with the help of *e(sig)* and *e(num)*.

Reviewing SSU parameter of CNF: This method doesn't influence the password choice of user and hence system interference and stress on memorability using this method become insignificant. This method is not much typo safe as generated honeywords contain closer numbers with respect to the number used in actual password of the user. Mounting DoS attack becomes easy in *CNF* as the generated honeywords are predictable by the adversary. Achieving high security standard in terms of addressing MSV and flatness criteria can be identified as major strengths of *CNF*. Like *modified-tail* and *caps-key*, the required storage cost is same as $k \times h$ bytes using *CNF* method.

3.4. Preprocessing technique

In [14], authors show that there are few specific digit patterns that can't be masked by *CNF* approach and this is where preprocessing may come handy. As an instance, repetition of digit d (> 2) for d times (identified as category 1) and repetition of digit d for $d^{/}$ times (identified as category 2) are examples of such patterns. In [14], authors proposed that if the digits used in user password fall under category 1, system generates same set of patterns for other digits (> 2). Thus, for original password as 333 the generated list of sweetwords for $k = 6$ may be as following–

55555 88888888 4444 **333** 666666 999999999

Whereas for the second category, if user password contains digits like 444 then the value of $d^{/}$ is considered as 3 and the generated list of probable sweetwords are as following for the same value of k as 6 –

555 **444** 888 222 111 666

Reviewing SSU parameter of the technique: This technique has no system interference and thus, imposes low stress on memorability. The occurrence of typo error is less probable here. Though this method sets high security standard in MSV resiliency and flatness, from security perspective, mounting DoS attack is easy for this approach. The maximum value of k becomes 7 for the passwords belonging to first category and that of $k = 9$ for the passwords belonging to the second category. Thus, the standard value of k ($= 20$) can't be achieved here.

4. Proposed Methodologies to Improve SSU Parameter

In this section, we propose few honeyword generation techniques that improve *SSU* parameter of some recently proposed approaches, mentioned in previous section. Here we propose four approaches namely – (a) Storage optimized *modified-tail* (*SOMT*) (b) Storage optimized *caps-key* (*SOCK*) (c) Storage optimized *close-number-formation* (*SOCNF*) and (d) Storage optimized *pre-processing* (*SOPP*) . Next we elaborate these approaches one by one.

4.1. SOMT: improving SSU parameter of modified-tail

To improve *SSU* parameter of *modified-tail*, the concept of *paired-distance-protocol* in [13] is used. While constructing S, we have set $|S| = k$. Now k characters are chosen in such a manner that avoids normal character selection tendency of user during formation of password. In Fig. 2, the probable set of characters that can be used to build the set S for $k = 20$ has shown.

@	/	}	2	(
T	r	&	j	?
;		+	L	*
∧	9	.	g	,

Fig. 2. Elements of set S of cardinality 20 in SOMT

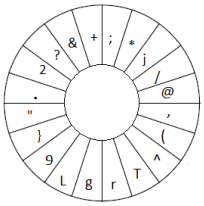

Fig. 3. Formed circular list using the elements of S

While selecting the *tail*, user chooses 2 different characters from the set S as of *modified-tail* approach. Then system calculates the paired distance (the number of cells that has to be traversed in clockwise direction to reach to an element from other element) between two consecutive elements of the *tail*, chosen by user. For example, if user selects his tail as '*Tg*' then the calculated paired distance based on the circular list shown in Fig. 3 will be 2. In password file F_P, system maintains this paired distance (2 here, may be in the hash format) along with the username and password. Instead of maintaining the original password index in the H_C, system maintains the first character of the *tail* chosen by user along with corresponding username.

If F_P is compromised, adversary obtains the password along with the paired distance. Now it is easy to check, for a particular paired distance, there are $|S|$ possible tails that can be formed. For example, if paired distance is 2 then possible tails can be formed starting from each element of circular list as – rL, $g9$, L} and so on. On the other hand, starting with a character, for a given paired distance, system always derives a unique *tail*. Thus, by only storing paired distance, system confused attackers among k possible options.

After retrieving the login information, system first derives paired distance from the submitted *tail*. If paired distance gets matched with the stored one then system directs the first character of the submitted *tail* to the H_C. If the stored character gets matched with the received character then H_C provides a positive feedback, otherwise a negative feedback is generated. Below we show, using *SOMT*, the contents of F_P and H_C for username u_i, password as p_i and selected *tail* as *Tg*.

F_P	<u_i, H(p_i), H(2)>	H_C	<u_i, T>

Analyzing SSU parameters of SOMT: As user has to remember a *tail* in *SOMT*, like in *modified-tail* thus, usability standard of the proposed approach remains same as *modified-tail* approach. Now without knowing the orientation of characters, adversary will not be able to know the paired distance, calculated from the *tail* chosen by user. Thus, without compromising F_P, the probability of mounting DoS attack can be calculated by following equation –

$$Pr(\text{DoS}) = (|S| - 1) \times \sum_{i=0}^{|\text{tail}|-1} \frac{1}{|S| - i} \tag{3}$$

For the standard values of $|S| = 20$ and $|tail|$ (length of the *tail*) as 2, the above probability can be evaluated as 0.05. Thus, *SOMT* makes *modified-tail* a strong security provider against DoS attack. The method is also typo safe as probability of entering a different *tail*, by typing mistake which evaluates the same paired distance as stored one, is also very less (in fact this too can be evaluated by using Equation 3). By sharing the circular list among different systems, as shown in[13], *SOMT* can also build robust security standard against MSV. Using *SOMT*, system only stores single extra information in terms of paired distance along with password. Thus, the value of k in this case only becomes 2.

4.2. SOCK: improving SSU parameter of caps-key

Using *SOCK* approach, system first retrieves the index of the capital letters – chosen by user, from the user's password. For example, if user chooses his password as *AniMal* then system retrieves 0 and 3 as the index values (starting with 0). System then converts all capital letters of user password in the small cases and stored the hashed value of it in F_P. The index values of capital letters are stored in the H_C server along with username. Thus, for username u_i and password selected as *AniMal* we show the content of F_P and H_C below –

F_P	<u_i, H(animal)>	H_C	<u_i, 0, 3 >

During login, user first submits his login information. System then retrieves the index of capital letters from submitted password string before converting it into small case. After converting the password in the small case, system makes a hash transformation of whole password string. If obtained hash value gets matched with the stored one then system directs the obtained index values of capital letters to the H_C server. If the index values get matched in the H_C then it sends a positive feedback signal to the system administrator, otherwise directs a negative feedback signal.

If we analyse the *SOCK* approach from the adversary point of view, then it is clearly noticeable that after inverting a password hash adversary gets confused among $\binom{n}{m}$ (n and m represent number of alphabets in password and how many of them are in capital letters, respectively) possible options (assuming that value of m is known to adversary). If the password information entered by the adversary gets matched with the stored string in lowercase but the indices of capital letters are wrongly identified by the adversary then system detects the attack and blocks that adversary.

Analyzing SSU parameters of SOCK: From usability perspective, *caps-key* and *SOCK* maintain same level of standard as user requires to remember same login information. *SOCK* attains low success rate while providing security against DoS attack but achieves high level of security standard while addressing the MSV issue. Using *SOCK*, attackers gets confused among $\binom{n}{m}$ number of sweetwords which may be much larger than $k = 20$ for $n \geq 6$ and $m = 2$. Thus, *SOCK* can be identified as stronger security provider than of *caps-key* in terms of achieving higher detection rate. We identify *SOCK* as *zero-storage-cost* honeyword based approach as, system doesn't have to store any extra information in F_P to lure the attackers. This differentiates *SOCK* from all other honeyword generation approaches, proposed so far.

4.3. SOCNF: improving SSU parameter of CNF

While optimizing the storage cost of *CNF*, we primarily motivated from[10], where authors argued that storing passwords in hash format is insignificant using a honeyword based approach as all honeyword based approaches play their role after adversary successfully inverts all password hashes. Using *SOCNF* system maintains the password in plain text format. The digits in the original user password are replaced with some other digit within a range $\pm r$, determined by the system administrator. Then system calculates the differences between the original digits used and the replaced digits in the password. System stores this difference in the H_C server. For example, for the original password chosen by user as *alex1992* system replaces 1992 by 1998, for the decided range by system administrator as ±20 (taken as example). System then calculates the difference as 6 and stores this information in H_C. Below we show the content of F_P and H_C for *SOCNF* for username u_i and password as *alex1992*.

F_p	< u_i, alex1998>	H_C	<u_i, 6>

Now while genuine user submits his password, system calculates the difference between digits in stored password and submitted password. If content of submitted password (excluding digits in it) matches with the stored one then

system directs the calculated difference to the H_C. If the difference gets matched then H_C sends a positive feedback. H_C sends a negative feedback if the difference value lies within $\pm r$ and doesn't get matched with the stored value. From the adversary point of view, the attacker gets confused among $2\times r$ possible options after obtaining the password, protected by *SOCNF* method.

Analyzing the SSU parameter of SOCNF: The security standard of *SOCNF* remains same as *CNF* approach. Likewise *SOCK*, *SOCNF* can also be identified as *zero-storage-cost* honeyword based approach. The usability parameters in *SOCNF* doesn't get differ with respect to the *CNF*.

Next we introduce *SOPP* which improves *SSU* parameter of preprocessing technique.

4.4. SOPP: improving SSU parameter of pre-processing

For both the categories of *pre-processing* approach – *category* 1 and *category* 2 (see Section 3.4), in *SOPP* we maintain a table in F_P, shown in Table 1.

Table 1. Maintained table in *SOPP* for generating decoys.

digit/index	0	1	2	3	4	5	6	7	8	9
Category 1	×	×	×	valid	valid	valid	valid	valid	valid	valid
Category 2	valid	valid	valid	valid	valid	valid	valid	valid	valid	valid

In *SOPP*, for *category 1*, system stores the password (may be in the hashed format or plain text) – after discarding the digits, along with the category information (here *category 1*). In the H_C, system stores the digit/index which has been repeated. For example, if user selects his password as *macro666666* then system stores *macro* (substring of password contains non-numeric data) along with *category 1* as user password info in F_P. As user uses digit 6 in his password thus, index value here will be 6 (see Table 1) and H_C will store 6 along with the username. We present content of F_P and H_C below for above example.

F_P:	<u_i, H(macro), category 1>	H_C:	<u_i,6>

From the attacker point of view, after obtaining compromised F_P, adversary gets to know the partial password along with the category information. But adversary wouldn't be able to understand the digit used by user. Entering the partial password (here *macro*) correctly without properly identified digit, helps H_C to detect the breach.

On the other hand for the password digits belonging to the second category, F_P stores the partial password information along with the category and the frequency of the digit. For example, for the password *macro5555*, F_P stores the password info as *macro, category 2* and 4 (as digit 5 repeats for 4 times). In H_C, system stores the digit 5 chosen by user along with username as shown below.

F_P:	<u_i, H(macro), 4, category2>	H_C:	<u_i,5>

When a user submits his login information, system first categorizes the nature of the digit string. If it belongs to *category 2* system then matches the partial password information along with frequency. If match occurs system directs the identified digit to the H_C which generates the positive feedback signal if the digit/index value matches otherwise, generates a negative feedback signal.

Analyzing SSU parameters of SOPP: From the security and usability perspectives *SOPP* maintains same standard as existing *pre-processing* technique. For the passwords belonging to *category 1*, system maintains single extra information as category. For *category 2* passwords, system maintains extra information as category and frequency, which reduce value of k by 7. Next, we give comparative analysis between existing honeyword methods and proposed approaches here.

Comparative study: We also compare the proposed storage optimized methods with the existing methods in[14] and some other recently proposed methods like, *take-a-tail*[10], *modeling-syntax*[12] and *chaffing-by-tweaking-digits* (CTD)[10]. The comparison results are based on *SSU* parameters and is shown in Table 2. The comparison result shows that the proposed approaches in this paper improve the security standard compared to existing *modified-tail* and *caps-key* approach. Also proposed approaches in this paper reduce the storage cost significantly over all other existing approaches.

Table 2. Comparative analysis among methods with respect to SSU parameter. "*" denotes conditional flatness (e.g. there exists no correlation between username and password). Bold fonts indicate the improvements using the proposed approaches.

Method	System-interference	Stress-on-mem	Typo-safety	DoS resiliency	Security against MSV	Flatness	Extra storage cost
modeling-syntax	no	low	yes	high	low	1/k *	k-1
take-a-tail	yes	high	yes	low	high	1/k	k-1
CTD	no	low	yes	high	low	1/k *	k-1
modified-tail	yes	low	yes	low	high	1/k	k-1
CNF	no	low	no	low	high	1/k *	k-1
caps-key	yes	low	no	low	moderate	1/k	k-1
pre-processing	no	low	yes	low	high	1/k	6 and 9
SOMT	yes	low	yes	**high**	high	1/k *	**1**
SOCNF	no	low	no	low	high	1/k *	**Null**
SOCK	yes	low	no	low	**high**	1/k	**Null**
SOPP	no	low	yes	low	high	1/k	**1** and **2**

5. Conclusion and Future Work

In this paper we have proposed few storage optimized honeyword based approaches which not only reduce the storage cost of previously proposed approaches but also improve the security standard from few aspects. We have also shown that in some cases, honeyword based model can be built without even storing any extra information in F_P. While improving the security and storage standards, we take necessary care not to reduce the usability standards of the proposed schemes compared to existing schemes. Improving the usability standards further of the proposed approaches without compromising any *SSU* parameter is our future plan. Nevertheless we believe that this paper reveals some important findings which encourage more use of honeyword based techniques in future.

6. Acknowledgments

This work is partially supported by a research grant from the Science & Engineering Research Board (SERB), Government of India, under sanctioned letter no. SB/FTP/ETA-226/2012.

References

1. J. Brainard, A. Juels, R. L. Rivest, M. Szydlo, M. Yung, Fourth-factor authentication: somebody you know, in: Proceedings of the 13th ACM conference on Computer and communications security, ACM, 2006, pp. 168–178.
2. D. Florencio, C. Herley, A large-scale study of web password habits, in: Proceedings of the 16th international conference on World Wide Web, ACM, 2007, pp. 657–666.
3. T. Kwon, S. Shin, S. Na, Covert attentional shoulder surfing: Human adversaries are more powerful than expected, Systems, Man, and Cybernetics: Systems, IEEE Transactions on 44 (6) (2014) 716–727.
4. A. van der Merwe, M. Loock, M. Dabrowski, Characteristics and responsibilities involved in a phishing attack, in: Proceedings of the 4th international symposium on Information and communication technologies, Trinity College Dublin, 2005, pp. 249–254.
5. J. Ma, W. Yang, M. Luo, N. Li, A study of probabilistic password models, in: Security and Privacy (SP), 2014 IEEE Symposium on, IEEE, 2014, pp. 689–704.
6. S. Designer, John the ripper password cracker [online document][cited 2008 oct 07], `http://www.openwall.com` (2008).
7. M. Weir, S. Aggarwal, B. De Medeiros, B. Glodek, Password cracking using probabilistic context-free grammars, in: Security and Privacy, 2009 30th IEEE Symposium on, IEEE, 2009, pp. 391–405.
8. C. Gaylord, Linkedin, last. fm, now yahoo? don't ignore news of a password breach, Christian Science Monitor 13.
9. D. Gross, 50 million compromised in evernote hack. *CNN* (March 2013).
10. A. Juels, R. L. Rivest, Honeywords: Making password-cracking detectable, in: Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security, ACM, 2013, pp. 145–160.
11. I. Erguler, Achieving flatness: Selecting the honeywords from existing user passwords.
12. H. Bojinov, E. Bursztein, X. Boyen, D. Boneh, Kamouflage: Loss-resistant password management, in: Computer Security–ESORICS 2010, Springer, 2010, pp. 286–302.
13. N. Chakraborty, S. Mondal, A new storage optimized honeyword generation approach for enhancing security and usability, arXiv preprint arXiv:1509.06094.
14. N. Chakraborty, S. Mondal, Few notes towards making honeyword system more secure and usable, in: Proceedings of the 8th International Conference on Security of Information and Networks, ACM, 2015, pp. 237–245.